VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizátor komplexních modelů tvořených pomocí NURBS

Visualization of Complex Models Represented by NURBS

2010

Tomáš Popek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Popek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vizualizátor komplexních modelů tvořených pomocí NURBS
Visualization of Complex Models Represented by NURBS

Zásady pro vypracování:

Seznamte se s reprezentací ořezaných NURBS a realizujte aplikaci, která bude schopna efektivně zobrazovat komplexní modely načítané z dodaného binárního souboru. Zobrazování realizujte nejlépe pomocí nativního OpenGL. Aplikace by měla zvládat práci s hladinami, výběr entit apod. Modelační nástroje nejsou požadovány.

1. Seznamte se s NURBS reprezentací ořezaných ploch.
2. Realizujte načítání modelu z dodaného souboru a jeho OpenGL a jeho rozšiřujících knihoven.
3. Vhodným způsobem odlište jednotlivé komponenty modelu pomocí materiálů (předpokládá se využití GLSL).
4. Vytvořte jednoduché UI pro správu zobrazeného modelu.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Fabián**

Datum zadání: 20.11.2009
Datum odevzdání: 07.05.2010

doc. Dr.Ing. Eduard Sojka
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Na tomto místě bych rád poděkoval všem, kteří mi s touto prací pomohli, protože bez nich by nikdy nevznikla. Obzvláště pak mému vedoucímu, díky kterému jsem tuto práci dovedl do zdárného konce.

Abstrakt

Tato bakalářská práce se zabývá vykreslováním komplexních modelů tvořených pomocí NURBS ploch. K tomuto vykreslování je využita knihovna OpenGL a její nádstavba GLU 1.3. Práce je rozdělena na část teoretickou a praktickou. V teoretické části jsou probrány důležité informace ohledně NURBS a jejich použití v OpenGL pomocí nádstavbové knihovny GLU. Dále je zde popsán soubor typu STEP, který pak využívá aplikace jako vstupní formát. V praktické části je probrána finální aplikace. V této části je detailně popsán postup od příjmu souboru až po jeho vykreslení v OpenGL. Fragmenty aplikace, kterými projde soubor se dají rozdělit na část parsovací, kde se ze souboru získají data potřebná pro tvorbu NURBS, část výpočetní, kde se provedou všechny důležité přepočty a část vykreslovací, která má za úkol smysluplnou reprezentaci získaných dat. Hlavním cílem této bakalářské práce je vytvořit aplikaci, která bude schopna výpočtu a vykreslení složitých scén uložených v souborech STEP, popřípadě CBA.

Klíčová slova: NURBS, STEP, CBA, parser, OpenGL, GLU

Abstract

This thesis deals with the rendering of complex models represented by NURBS surfaces. For this rendering OpenGL and its superstructure GLU 1.3 are used. The thesis is divided into theoretical and practical part. The theoretical part discusses important information about NURBS and their use in OpenGL, using the add-on library GLU. Then STEP file, which is used as input format for the application is described. In the practical part final application is discussed. This section describes in detail the process from receiving the file to its rendering in OpenGL. Fragments of an application, which file passes, can be split into three parts: parsing part, where we get the necessary data for creating NURBS, computational part, where all the necessary calculations are performed and rendering part, which is responsible for a meaningful representation of the obtained data. The main objective of this thesis is to create an application capable of calculating and rendering of complex scenes stored in files such as STEP, or CBA.

Keywords: NURBS, STEP, CBA, parser, OpenGL, GLU

Seznam použitých zkratek a symbolů

ACIS	–	Více možných překladů. Nejpravděpodobnější je: Alan, Charles, Ian's System
GLU	–	OpenGL Utility Library
IGES	–	Initial Graphics Exchange Set
NURBS	–	Non-Uniform Rational Basis Spline
OpenGL	–	Open Graphics Library
STEP	–	STandard for Exchange od Product model data

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Hlavní náplní samotné práce bylo vytvořit parser, který dokáže rozložit soubory typu STEP na jednotlivé entity. Tyto entity je pak důležité rozložit na dílčí fragmenty, které dokážeme přepočíst na atributy potřebné k tvorbě NURBS ploch a NURBS křivek. Takto získané křivky a plochy bude potřeba sloučit do jedné komplexní scény, kterou následně vykreslíme pomocí OpenGL.

Aplikace byla vytvořena za účelem rozšíření funkcionality stávajícího programu pro další typ souboru. Soubory, které je aplikace schopna zpracovávat, jsou formátu STEP a CBA. V textu práce budou průběžně poskytnuty základní informace potřebné k pochopení NURBS ploch a křivek. Tyto informace nám dají základ pro pochopení aplikačních tříd v dalších částech práce. Dále zde probereme historii a základy týkající se souboru formátu STEP. Z tohoto popisu si budeme schopni představit hierarchii samotného souboru potřebnou k jeho rozložení. Dále je zde obsažen stručný popis kniho- vny OpenGL. Díky této knihovně budeme schopni programově vykreslovat hotové plochy i s jejich ořezy.

V další části dokumentu věnované výsledné aplikaci bude probráno parsování dat ze vstupního souboru. Dále zde bude popis a vysvětlení tvorby samotných instancí entit získaných z těchto souborů. Tyto instance bude potřeba podrobit několika převodním výpočtům, aby byla zajištěna validita a kompatibilita získaných dat s původní aplikací. Takto transformovaná data budeme nakonec schopni vykreslit do finální scény, kterou budeme moci ovládat pomocí klávesnice a myši. Závěrem je shrnut výsledek aplikace, možnosti použití a nápady na její budoucí vylepšení. V poslední části dokumentu nalezneme reference vhodné k hlubšímu pochopení problematiky a také přílohy s příklady některých důležitých částí.

2 NURBS

Zkratkou NURBS jsou v počítačové grafice označovány jedny z nejpoužívanějších typů parametrických křivek a ploch. Tato zkratka je v této práci využívána v těchto kontextech:

- **NURBS křivky** — Racionální neuniformní B-splajny. Jedná se o zobecnění neracionálních B-splajnů o racionalitu (váhy řídicích bodů) a uzlový vektor (knot vector).
- **NURBS plochy** —- Plochy tvořeny pomocí výše zmíněných B-splajnů. Vychází z podobného matematického modelu jako křivky[9].

Vysvětlení jednotlivých slov ve zkratce NURBS:

- **Non-Uniform** — Odkazuje na velkou flexibilitu tvorby uzlového vektoru, který může mít narozdíl od uniformních křivek jednotlivé uzly různě vzdáleny od sebe.
- **Rational** —- Využívá posloupnosti vah kontrolních bodů. Díky těmto vahám se snadněji ovlivňuje tvar tvořené křivky.
- **B-Splines** —- Konec zkratky značí, že NURBS mnoho svých vlastností dědí ze starší generace uniformncích bázových křivek[9].

NURBS je matematický model převážně používaný v počítačové grafice pro konstrukci volných tvarů. Volné tvary, nebo také obecné plochy, jsou hojně využívány v CAD systémech. Tyto plochy nejsou standardně tvořeny body, které by jinak určovaly jejich hrany, zlomy a celkový tvar. Právě díky využití křivek namísto bodů je kostrukce takovýchto ploch snazší a výsledné plochy získávají mnohem větší hladkost. NURBS jsou dnes součástí mnoha ISO standardů, jako například IGES, STEP nebo ACIS a využívá je nespočet modelovacích programů jako Maya, Cinema4D, Rhino3D, CATIA a podobné. Důvody, proč je tyto programy využívají, jsou:

- Poskytují flexibilitu při modelování velkého množství různých tvarů.
- Relativně vysoké rychlosti výpočtů, díky stabilním algoritmům.
- Nízké nároky na místo v paměti pro uložení.
- Rychlé přepočty změn při konstrukci křivek.
- Při transformaci NURBS se pouze aplikuje transformační matice na kontrolní body, čímž se značně urychlí výpočet.

NURBS tedy své využití nalézá ve strojírenství, loďařství, automobilovém průmyslu, letectví a filmových animacích[4].

2.1 NURBS křivky

NURBS křivky jsou definovány uzlovým vektorem, stupněm, polem kontrolních bodů a jejich vah.

2.1.1 Kontrolní body

Kontrolní body určují tvar křivky v prostoru. Každý takový bod se skládá ze čtyř složek. Tyto složky jsou X, Y, Z a W, kde první tři standardně určují polohu bodu v prostoru a poslední je hodnota váhy. Váhu si můžeme představit jako sílu magnetu, který přitahuje část křivky k tomuto bodu tak, že čím větší váha, tím větší síla magnetu. Váha každého bodu může nabývat pouze nenulových hodnot. Poslední dobou se ale od použití vah upouští, jelikož práce s nimi se při programování nemusí obejít bez problémů, respektive se váhy berou s hodnotou 1. Existuje speciální případ kontrolního bodu, který má váhu rovnu nule. Takový bod se nazývá nevlastní řídící bod. S těmito řídícími body však nepracuji a nebudu je dále rozebírat. Stupeň křivky pak na těchto bodech zaručuje její hladkost.

2.1.2 Uzlový vektor

Uzlový vektor je posloupnost hodnot, které určují, kde a jak ovlivní kontrolní body tvar křivky. Počet uzlů vektoru je roven součtu počtu kontrolních bodů a stupně křivky + 1. Uzlový vektor rozděluje parametrický prostor na dílčí intervaly nazývané knot-span. V tomto vektoru je důležité dbát na neklesající seřazení hodnot. Uzly mohou nabývat různých hodnot, avšak při výpočtech a práci s vektorem je vhodné je normalizovat do intervalu $\langle 0, 1\rangle$. Normalizace uzlového vektoru je podrobně probrána v kapitole 5.4.1. Kromě správného seřazení uzlů je také nutné dávat si pozor na multiplicity hodnot. Počet povolených multiplicit sousedních uzlů je roven stupni křivky. Samotné hodnoty uzlů jsou nepodstatné, důležité jsou intervaly, které tvoří jednotlivé dvojice. Práce s uzlovým vektorem tedy závisí na poměrech velikostí krajních hodnot každého intervalu. Například vektory $(0, 0, 1, 2, 3, 3)$ a $(0, 0, 2, 4, 6, 6)$ udávají stejný tvar křivky, protože poměry stran jednotlivých dvojic jsou totožné. Dále je důležitý i řád křivky, který není nic jiného než stupeň křivky + 1. Nejčastěji používané křivky jsou řádu 3 a 4. Řád nám určuje, jaký maximální počet nejbližších kontrolních bodů má vliv na bod ležící na křivce[15].

2.1.3 Matematická definice NURBS

Mějme $m + 1$ kontrolních bodů P_i, $m + 1$ kladných reálných čísel w_i, stupeň křivky n a uzlový vektor $t = (t_0, t_1, \cdots, t_{n+m+1})$.
Definice:

$$C(t) = \frac{\sum_{i=0}^{m} w_i P_i N_i^n(t)}{\sum_{i=0}^{m} w_i N_i^n(t)}$$

kde:

$$t \in \langle t_n, t_{m+1}\rangle$$

Bázová funkce N_i^n je definována rekurentně:

Nechť $t = (t_0, t_1, \cdots, t_{n+m+1})$ je uzlový vektor. B-spline funkce stupně n je pak definována jako:

$$N_i^0(t) = \begin{cases} 1 \text{ pro } t \in \langle t_i, t_i + 1) \\ 0 \text{ jinak} \end{cases}$$

$$N_i^n(t) = \frac{t - t_i}{t_{i+n} - t_i} N_i^{n-1}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} N_{i+1}^{n-1}(t)$$

kde:

$$0 \leq i \leq s - n - 1, 1 \leq n \leq s - 1, \frac{0}{0} := 0$$

Za každou nurbs křivkou či plochou se ukrývají tyto základní rovnice. Hlavním důvodem složitosti vzorců je to, že pro každý parametr t se vypočítají jiné bázové polynomy, na rozdíl od Bezierových či Coonsových křivek[4]. Tímto jsme si definovali NURBS křivky a můžeme přejít k plochám.

2.2 NURBS plochy

NURBS plochy se tvoří jako tenzorový součin těchto křivek, kde jedna reprezentuje směr u a druhá v. Jedna plocha má tedy dva stupně, množinu kontrolních bodů a dva uzlové vektory. Všechny tyto atributy musí splňovat stejné požadavky jako u křivek. To znamená, že počet uzlů v uzlových vektorech je závislý na počtu kontrolních bodů a stupni pro daný směr v parametrickém prostoru. Hodnoty, kterých smí uzly nabývat, jsou opět v neklesající posloupnosti a mají povolené multiplicity podle stupně křivky. Díky využití uzlového vektoru a vah kontrolních bodů je možné, na jinak hladké ploše, vymodelovat i ostré zlomy, což je využitelné například při modelování již zmiňovaných karosérií a lodí. Pro plochu se stupni p a q s uzlovými vektory u a v je definice podobná jako u křivek:

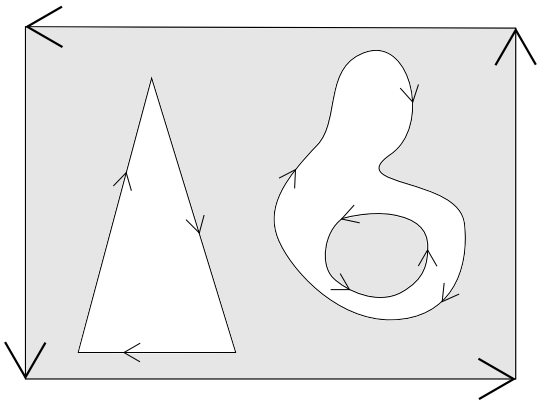
$$S(u, v) = \frac{\sum_{i=0}^{m} \sum_{j=0}^{n} N_{i,p}(u) N_{j,q}(v) w_{i,j} P_{i,j}}{\sum_{i=0}^{m} \sum_{j=0}^{n} N_{i,p}(u) N_{j,q}(v) w_{i,j}}$$

2.2.1 Ořezové křivky - trimming curves

Vzhledem k obdelníkovému tvaru parametrické plochy, nad kterou je NURBS definován, by využití takovýchto ploch, pokud by už dále nešly upravit, bylo poměrně malé. Proto lze tyto plochy upravovat pomocí ořezových křivek. Pomocí ořezávání ploch lze snadno vytvořit spoustu složitých tvarů, které bychom jinak modelovali velmi těžko. Ořezávání se provádí pomocí ořezových křivek (trimming curves), které jsou zadané v parametrickém prostoru pomocí parametrů u a v. Tyto křivky vlastně určují, které části ploch se nebudou vykreslovat.

Ořezávání se provádí tím způsbem, že uzlové vektory ořezávané plochy znormalizujeme do intervalu $\langle 0, 1 \rangle$. Díky této normalizaci je nyní naše plocha omezena čtvercem,

který má vrcholy (0,0) až (1,1). Normalizace ploch a křivek není povinná, jedná se o ekvivalentní zápis, ale pro práci s nimi bývá zvykem ji provést. Pokud tedy není zadána žádná ořezová křivka (trimming curve), pak se plocha vykreslí celá. Máme-li zadané ořezové křivky, pak se naše plocha rozdělí na části, které se vykreslovat budou, a na ty, které ne. Atribut, jak poznat, že daná oblast, kterou křivka ohraničuje, zůstane viditelná, nebo ne, je její orientace. Orientací máme na mysli směr vykreslování jednotlivých řídících bodů. Takže, pokud je křivka orientována ve směru hodinových ručiček (CW), pak se daná oblast vykreslovat nebude. Orientace křivky proti směru hodinových ručiček (CCW) znamená, že křivka ohraničuje oblast, která má být zachována[14].



Obrázek 1: Ukázka závislostí ořezů na orientaci

Problémy plynoucí z ořezávání ploch pomocí křivek jsou různé. Nejčastěji se vyskytuje problém s převodem z datového typu double na float. Tento problém vychází z různých počtů čísel za desetinnou čárkou. Při ořezu double čísla na float se pak často stává, že ztratíme ta místa, která uchovávala pro nás důležité hodnoty a díky tomu vznikají nechtěné duplicity. Další problém s těmito typy je například ten, že když porovnáváme hodnoty float a double mezi sebou, tak musíme ošetřit, po jak vzdálené desetinné místo hodláme porovnávat. Bez tohoto ošetření by se pak mohlo často stávat, že získáme negativní výsledek, i když porovnáváme dvě zdánlivě stejná čísla, která se liší na, pro nás nedůležitém, n-tém desetinném místě. Dále je zde častý problém s komplikovanou teselací ořezaných ploch, nebo s display listy, které nemusí vykreslovat všechny plochy, i když jsou validní. Největším problémem v této práci byl komplikovaný převod modelově zadaných křivek na jejich parametrické vyjádření pomocí složitých, nepřesných a časově náročných výpočtů. Většinu výše popsaných problémů proberu hlouběji v další části této práce věnované programové dokumentaci.

3 STEP

STEP je mezinárodní standard pro přenos dat. Soubory tohoto typu využívají normu ISO 10303, jejíž cílem je poskytovat data v průběhu výroby produktu nezávisle na platformě. STEP je vlastně řada mezinárodních standardů vystavěných kolem specifických aplikačních protokolů a generických zdrojů. Právě soubory tohoto typu se hojně využívají při práci s NURBS díky jejich jednoduchosti a stabilitě. Nicméně mnoho CAD systémů na tento formát zanevřelo, protože některé překladače nedokázaly správně z těchto souborů vyjmout data. Díky těmto chybným čtením pak vznikaly různé odchylky v modelových plochách, které se pak musely náročně a zdlouhavě opravovat. Tyto problémy se však týkaly hlavně složitějších ploch a jejich ořezů. K tomuto problému se dostanu později. Samotný STEP je organizovaný jako soubor jednotlivých částí, kde každá část je prezentována samostatně. Tyto části spadají pod jednu z následujících skupin[3]:

- description methods
- integrated resources
- application protocols
- abstract test suites
- implementation methods
- conformance testing

3.1 STEP-file

STEP-file je nejrozšířenějším využitím STEPu. Díky jeho ASCII struktuře je snadno čitelný a překladatelný. První verze tohoto formátu spatřila světlo světa v roce 1994. Rokem 2002 byla tato verze kompletně opravena, zbavena mnoha chyb a rozšířena o pár nových sekcí. Formát, který definuje STEP-file, má označení ISO10303-21. Tento formát definuje zápis, strukturu a reprezentaci dat. Přípony souboru, které využívají tento standard, jsou $.stp$ a $.step$. Samozřejmě díky využití onoho ASCII můžeme mít STEP i v prostém $.txt$ souboru. Samotný soubor se dá rozdělit na dvě části, a to na hlavičku a datovou sekci. Jak je vidět na ukázce v příloze B, tak soubor využívá různá klíčová slova. Celý soubor je ohraničen ISO-10303-21 a END-ISO-10303-21. Tyto dva tagy nám určují začátek a konec souboru. Dále jsou zde již zmiňované sekce HEADER a DATA. Obě tyto sekce jsou ukončeny tagem ENDSEC[18].

3.1.1 Hlavičková sekce

Hlavička obsahuje několik podskupin, které popisují, co a jak soubor reprezentuje. Počet a jména podskupin závisí na verzi STEP-file. Ty nejdůležitější podskupiny jsou:
FILE_DESCRIPTION

- **description** — Uživatelem zadaný popis souboru. Může zůstat nevyplněný.

- **implementation_level** — Verze step souboru.

FILE_NAME

- **name** — Název souboru.
- **time_stamp** — Čas vzniku.
- **author** — Jméno a e-mail autora.
- **organization** — Název organizace.
- **preprocesor_version** — Název a verze systému, ve kterém byl soubor vytvořen.
- **originating_system** — Původní systém, ve kterém byl soubor vytvořen.
- **authorization** — Jméno a e-mail osoby, která soubor autorizovala.

FILE_SCHEMA — Použité schéma. Závisí na verzi STEPu.

3.1.2 Datová sekce

Tato sekce obsahuje data, která jsou uložena v závislosti na použitém schématu. Přístup k těmto datům je pak velice jednoduchý, jelikož se využívá různých klíčových slov a odkazů. Obecný záznam dat v této sekci se dá popsat takto:

- Každý záznam má své klíčové slovo, které určuje strukturu jeho zápisu.
- Každý záznam má jednoznačný identifikační klíč, který je zapsán ve formátu #123=.
- Zápis polí, listů a jiných seznamů je v jednoduchých závorkách, kde jsou jednotlivé prvky odděleny čárkou.
- Pokud je v atributu obsažen řetězec znaků, jako například název, pak je umístěn v jednoduchých uvozovkách.
- Je-li název prázdný, pak se na jeho místo dává znak $.
- Jestliže záznam využívá jiných záznamů, pak jsou odkazy na tyto entity psány ve formátu #1234.
- Booleovské hodnoty jsou zapsány mezi tečky (.T. nebo .FALSE.).
- Konec záznamu se značí středníkem.
- Float a double hodnoty využívají desetinných teček.
- Nedeklarovaný odkaz využívá místo klíče #123 znak hvězdičky *.

Obecným příkladem záznamu je například toto:
#128=NÁZEV_ENTITY('jméno',(prvek,prvek,prvek),.T.,#odkaz);

Další práci s datovou sekcí a popis jednotlivých slov, které využívá můj program, podrobněji proberu v části věnované parseru.

4 OpenGL

OpenGL neboli Open Graphics Library je standard, který definuje jazyk pro tvorbu aplikací pracujících s 2D a 3D grafikou. Tento jazyk obsahuje přes 250 různých funkcí, které mohou být použity pro vykreslení komplexních 3D objektů a scén. OpenGL bylo vytvořeno společností Silicon Graphics Inc. v roce 1992 a od té doby je široce používáno v CAD systémech, herním průmyslu nebo různých simulacích. Současným konkurentem této knihovny je obdobná knihovna Direct3D. OpenGL slouží hlavně pro:

- Skrytí složitosti rozhraní různých grafických akcelerátorů.
- Sjednocení možností různých platforem pod jeden celek.

Základní funkce OpenGL je příjmout matematicky vyjádřená primitiva jako body, nebo polygony a převést je do srozumitelné interpretace na monitor pomocí pixelů. Díky nízkoúrovňovému programování v OpenGL je pořeba, aby programátor definoval každý krok, kerý se provede pro vykreslení scény. To programátorovi dává velkou škálu možností tvorby vykreslovacích algoritmů. Popis procesů prováděných při zpracovávání:

- Vyhodnocení polynomiálních funkcí, které definují určité vstupy jako NURBS plochy a jejich aproximace.
- Vertexové operace. Transformace a osvětlení scény se provádí v závislosti na použitém materiálu.
- Ořezávání objektů mimo viditelnou část scény pro urychlení vykreslování.
- Rasterizace původních primitiv do pixelů.
- Ve finální fázi jsou fragmenty uloženy do Frame bufferu.

Obecně se většina těles v OpenGL tvoří pomocí základních primitiv jako jsou trojúhelníky, ze kterých se pak sestaví nepravidelná trojúhelníková síť (TIN). Co ale dělat potřebujeme-li vymodelovat objekty, u kterých dbáme na přesnost povrchů, zachování topologie těles, nebo snadnější manipulaci při animacích? V tomto případě sáhneme po NURBS křivkách a plochách, které dokáží utvářet hladké, přesné povrchy nebo umožnují snadnou manipulaci díky použití kontrolních bodů, vah a uzlových vektorů[17].

4.1 Nádstavbová knihovna GLU

V průběhu této dokumentace je často zmiňována knihovna GLU. Tato knihovna je rozšíření standardního OpenGL, které neobsahuje mnoho důležitých funkcí a příkazů. Důvodem, proč tyto příkazy v OpenGL chybí, může být například jejich složitost a malá použitelnost. GLU tedy rozšiřuje možnosti práce s OpenGL. Pro nás důležitá fakta o této knihovně jsou, že právě v ní jsou uloženy funkce důležité pro práci a manipulaci s NURBS a hlavně pomocné funkce pro tessalaci. Pro nás užitečné funkce obsahuje tato knihovna verze 1.3. Například bez výše zmiňované tessalace bychom nebyli schopni správného

ukládání výsledných ploch do display listů, ani jejich načítání. Toto by mělo za následek znatelné zpomalení aplikace až k hranici použitelnosti. Rovněž jsou v této knihovně zakomponovány různé inicializační funkce, funkce pro kontrolu chyb a zjišťování jejich příčin za běhu aplikace. Knihovna GLU obsahuje i velmi užitečné funkce pro práci s bitmapami a možnostmi jejich využití jako textur. Jak bylo řečeno výše tak s touto knihovnou je spojena tessalace NURBS ploch. Tessalace ploch znamená její převedení na trojúhelníky (trojúhelníky jsou standardní primitiva používaná v OpenGL).

4.2 Inicializace okna a scény v OpenGL

Před samotným vykreslováním scény a modelů je důležité definovat si základní vlastnosti okna, které budeme využívat v průběhu práce s ním. Tyto vlastnosti jsou:

- Velikost okna.
- Název okna.
- Světlo, materiály a vlastnosti odlesků použité pro tvorbu modelů.
- Ořezávací rovina, která určuje vzdálenost vykreslování modelů.
- Pokud je v plánu přidat ovládání k manipulaci se scénou, pak je vhodné zde definovat i pomocné proměnné, které budou využívány při výpočtech a překreslování.
- V tomto případě, kdy pracujeme s NURBS plochou, zde definujeme objekt této plochy.

4.3 Křivky v OpenGL

Postup tvorby NURBS křivek lze shrnout do několika bodů:

- Nejprve je vhodné korektně nastavit stavový stroj OpenGL. Vzhledem k tomu, že se při vykreslování NURBS generuje velký počet vrcholů, je možné zvážit, zda se má pro každý vrchol automaticky počítat i jeho normála. Pokud to není nutné (například se nepoužívá osvětlení ani odlesky), je možné výpočet normálových vektorů zakázat funkcí $glDisable(GL\_AUTO\_NORMAL)$, opětovné povolení se provede funkcí $glEnable(GL\_AUTO\_NORMAL)$.
- Dále se musí vytvořit programový objekt, který reprezentuje zvolenou NURBS křivku v operační paměti počítače. Vytvoření instance tohoto objektu se provede pomocí funkce $gluNewNurbsRenderer()$.
- Pro NURBS křivky je možné nastavit několik atributů. Tyto atributy se dají nastavit pomocí funkce $gluNurbsProperty()$.
- Následně je možné provést registraci callback funkce, jež se zavolá v případě výskytu nějaké chyby při vytváření či renderování NURBS křivek. Pro registraci je zapotřebí zavolat funkci $gluNurbsCallback()$. Díky této funkci se na příkazové řádce smysluplně reprezentují případné chyby, což nám dává větší možnosti při ladění.

- Počátek vytváření NURBS křivky (tj. zadávání řídících bodů a uzlového vektoru) se specifikuje zavoláním funkce $gluBeginCurve()$.
- Poté je již možné specifikovat jednotlivé řídící body a složky uzlového vektoru přes funkci $gluNurbsCurve()$.
- Konec specifikace parametrů NURBS křivky, a tím i ukončení jejího vykreslování, je indikován zavoláním funkce $gluEndCurve()$.
- Po použití NURBS křivky je dokonce možné zrušit i její objekt zavoláním funkce $gluDeleteNurbsRenderer()$[11].

Ukázky použití jednotlivých příkazů jsou uvedeny níže.

```
GLUnurbs *nurbs;                    // objekt NURBS plochy
nurbs = gluNewNurbsRenderer();      // vytvoreni NURBS

// registrace Error Callback funkce
gluNurbsCallback( nurbs, GLU_ERROR, ( _GLUfuncptr )nurbsErrorCallback );
// definovani property pro tessalaci ploch
gluNurbsProperty( nurbs, GLU_NURBS_MODE, GLU_NURBS_TESSELLATOR );
```

Výpis 1: Ukázky příkazů pro inicializaci NURBS

Před samotným vykreslením NURBS křivky je vhodné si definovat všechny její důležité atributy, které budeme potřebovat. Těmito atributy jsou:

Název	**Popis**
nurbs	Obsahuje objekt NURBS křivky, který se bude měnit.
knotsCount	Počet uzlů v uzlovém vektoru.
&knots[0]	Ukazatel na první prvek pole uzlového vektoru.
stride	Velikost kroku. Po převedení dvou rozměrného pole kontrolních bodů na jednorozměrné, krok určuje vzdálenost jednotlivých prvků v tomto poli. Například máme-li pole s prvky $[x_1, y_1, z_1, x_2, y_2, z_2, x_n, y_n, z_n]$, pak vzdálenost jednotlivých prvků je 3.
&points[0]	Ukazatel na první prvek pole kontrolních bodů.
order	Řád křivky (stupeň křivky+1).
type	Určuje typ křivky. Pro vykreslení NURBS křivky jsou možné hodnoty pouze GL_MAP1_VERTEX_3, pro body určené jako $P = (x, y, z)$ a GL_MAP1_VERTEX_4, pro body definované jako $P = (x, y, z, w)$ (v tomto případě by byla i hodnota $stride = 4$).

Tabulka 1: Atributy NURBS křivky

Po definici těchto atributů se pak vykreslení křivky provádí takto:

```
gluBeginCurve(nurbs);
    gluNurbsCurve(nurbs,knotsCount, &knots[0],stride,&points[0],order,type);
gluEndCurve(nurbs);
```

Výpis 2: Vykreslení NURBS křivky

4.4 Plochy v OpenGL

Podobně jako křivky, se pak v OpenGL tvoří i NURBS plochy. Stejně jako u křivky je potřeba zajistit objekt, který bude reprezentovat NURBS plochu. Volání metod pro nastavení properties i callback funkcí zůstává nezměněné. Jediným rozdílem jsou názvy některých příkazů používaných při definici samotné plochy a počet jejich atributů.
Tyto nové atributy jsou:

Název	**Popis**
knotsCountU	Počet uzlů v uzlovém vektoru pro směr u.
&knotsU[0]	Ukazatel na první prvek pole uzlového vektoru pro směr u.
knotsCountV	Počet uzlů v uzlovém vektoru pro směr v.
&knotsV[0]	Ukazatel na první prvek pole uzlového vektoru pro směr v.
strideU	Krok v poli kontrolních bodů pro směr u.
strideV	Krok v poli kontrolních bodů pro směr v.
orderU	Stupeň plochy + 1 pro směr u.
orderV	Stupeň plochy + 1 pro směr v.

Tabulka 2: Atributy NURBS plochy

Po definici těchto atributů se pak vykreslení plochy provádí takto:

```
gluBeginSurface(nurbs);
      gluNurbsSurface(nurbs,
            knotsCountU, &knotsU[0],
            knotsCountV, &knotsV[0],
            strideU,strideV,
            &points[0],
            orderU, orderV,
            GL_MAP2_VERTEX_3);
gluEndSurface(nurbs);
```

Výpis 3: Vykreslení NURBS plochy

Výše uvedený příkaz $gluNurbsProperty(object, property, value)$ se dá využít i pro nastavení způsobu tessalace vykreslovaných ploch. Atributy tohoto příkazu budou nastaveny na $gluNurbsProperty(nurbs, GLU\_SAMPLING\_METHOD, sampling)$, kde atribut sampling může nabývat hodnot:

- **GLU_PATH_LENGTH** — Určuje, že vykreslené plochy s maximální délkou v pixelech, nejsou větší než hodnota specifikována v GLU_SAMPLING_TOLERANCE. Výchozí hodnota.
- **GLU_PARAMETRIC_ERROR** — Definuje, že při vykreslování plochy bude hodnota uložena v GLU_PARAMETRIC_TOLERANCE určovat maximální vzdálenost v pixelech mezi polygony a plochou, na které se tessalace provádí.
- **GLU_DOMAIN_DISTANCE** — Určuje šířku vzorkování pro směry u a v.

4.5 Ořezy plochy pomocí křivek

Postup tvorby ořezových křivek je prakticky stejný, jako u tvorby běžných NURBS křivek. Jediným rozdílem je při volání takovéto křivky její typ a také název tohoto volání. Možné typy ořezových křivek jsou tyto:

- **GLU_MAP1_TRIM_2** — Určuje typ ořezové křivky určené body $P = (u, v)$.
- **GLU_MAP1_TRIM_3** — Určuje typ ořezové křivky určené body $P = (u, v, w)$.

Volání ořezové křivky na ploše se musí provádět mezi příkazy $gluBeginSurface(nurbs)$ a $gluEndSurface(nurbs)$, aby bylo jednoznačně určeno, kterou plochu bude křivka ořezávat. Narozdíl od obyčejné NURBS křivky, kterou jsme chtěli pouze vykreslit, u této křivky záleží na její orientaci. Smysl orientace jsme si vysvětlili v předchozí kapitole 2.2.1 věnované NURBS plochám a jejich ořezům.

```
gluBeginSurface(nurbs);
gluNurbsSurface(nurbs,
                knotsCountU, &knotsU[0],
                knotsCountV, &knotsV[0],
                strideU,strideV,
                &points[0],
                orderU, orderV,
                GL_MAP2_VERTEX_3);

gluBeginTrim(nurbs);
   gluNurbsCurve(nurbs,knotsCount, &knots[0],stride,&points[0],order,type);
gluEndTrim(nurbs);
gluEndSurface(nurbs);
```

Výpis 4: Ořez plochy pomocí NURBS křivky

Pokud je potřeba ořezovou křivku složit z více otevřených křivek, pak mezi příkazy $gluBeginTrim(nurbs)$; a $gluBeginTrim(nurbs)$; vloží příkaz $gluNurbsCurve(\ldots)$ pro každou z nich. Důležité je dbát na správné pořadí těchto otevřených křivek tak, aby na sebe navazovaly podle počátečních a koncových bodů. Toto navazování se v praxi provádí „svařováním“ křivek, které kontroluje možné návaznosti křivek právě podle jejich kontrolních bodů. Svařování je hlouběji probráno v programové části dokumentace v kapitole 5.6.1.

5 Programová část

Tato aplikace byla vytvořena za účelem rozšíření funkcionality již stávající aplikace mého vedoucího. Tudíž bylo v pozdější části programu důležité, aby bylo využíváno stejných datových typů a metod. Aplikace, kterou jsem naprogramoval, se skládá z několika částí:

- **parser** — V této části aplikace načte a výjme ze souboru jen pro nás důležité entity, se kterými budeme pracovat.
- **tvorba entit** — Každá entita má vlastní způsob rozparsování, který se v této části zavolá a díky provázanosti těchto entit se vytvoří „stromová struktura“, kterou popíšu v sekci 5.2.
- **converter** — V této části se provádí převod mých Step-tříd na obecné třídy, které bude moci využívat i původní aplikace bez potřeby znalosti formátu Step.
- **přepočty a normalizace** — Tato část se prolíná vzájemně s converterem a provádí se zde různé převody a normalizace křivek i ploch.
- **vykreslení universa** — Universum je soubor všech objektů ve scéně, které se zde vykreslí pomocí knihovny GLUT a OpenGL.

5.1 Parser

Práce třídy Parser spočívá v otevření vstupního souboru, rozpoznání začátku pro nás důležitých dat a jejich uložení do mapy. Data jsou v mapě uložena pod číselným klíčem a mají hodnotu typu string. Klíč těchto dat je jedinečný a je stejný jako číslo entity ve Step-file. Hodnota těchto dat je pak textový řetězec, který je v souboru umístěn za klíčem.

Máme-li například podobný záznam:
#128=NÁZEV_ENTITY('jméno',(prvek,prvek,prvek),.T.,#odkaz);

Pak jeho klíčem v mapě bude číslo **128** a hodnotou řetězec
NÁZEV_ENTITY('jméno',(prvek,prvek,prvek),.T.,#odkaz)

Způsob, jakým Parser určuje, která entita je pro nás potřebná a která ne, je prostý. Byl vytvořen buffer, který se bude plnit číselnými klíči všech důležitých entit. Při nalezení a uložení první, pro nás důležité, entity do mapy, se prochází její řetězec a postupně se z něj separují další důležité klíče ve tvaru **#odkaz**. Nalezené odkazy se ukládají do bufferu a při dalším postupném průchodu souborem se kontroluje, zdali je klíč stávající entity obsažen uvnitř bufferu. Pokud je nový klíč obsažen v bufferu, pak se jedná o důležitou entitu, která se uloží do mapy a opět se projde její řetězec, aby se vyseparovaly další odkazy pro buffer. První entitu, nebo skupinu entit, která určuje počátek struktury, a od které se odvíjí další a další záznamy, najdeme podle názvu. První STEP entita má název **SHELL_BASED_SURFACE_MODEL** a vždy zastává jeden nebo více 3D modelů. Tímto způsobem dokážeme projít celý soubor, tak abychom do paměti uložili pouze entity,

které později budou využity při tvorbě instancí vlastních tříd. Třída Parser, po načtení souboru a jeho rozparsování do mapy, vrací mapu, která obsahuje klíče a jejich hodnoty, a také pole s čísly klíčů pro počátky modelových struktur. Využití této třídy nám zaručí, že se v paměti nebude zabírat příliš velké místo hodnotami, které pro nás nemají žádný užitek. V malých souborech by velikost „neužitečných“ hodnot byla zanedbatelná, avšak při práci s většími bloky dat by se bez této selekce jednalo o výrazně větší využití paměti.

5.2 Tvorba entit

Po rozparsování souboru a vytvoření mapy parserem se přejde k druhé fázi zpracování dat, a to k vytvoření instancí tříd pro každou entitu. Každá třída reprezentující Step-file entitu obsahuje konstruktor, který jako parametry přijímá klíč konkrétní entity a výše zmiňovanou mapu. Tento konstruktor z mapy vyjme hodnotu podle klíče a zavolá svůj třídní parser, který načtený řetězec rozparsuje na požadované atributy. Pokud je v řetězci, jako jeden z atributů, obsažen další odkaz na novou entitu, pak se při parsování tohoto odkazu vytváří nová instance této entity a rekurzivně se vytvoří kompletní provázaná struktura. Ukázka těchto tříd a jejich propojení je znázorněna pomocí UML diagramu v příloze (obrázek A).

5.2.1 Shell_Based_Surface_Model a Open_Shell

Tyto dvě entity se od sebe prakticky neliší. Díky nim se znázorňuje, které „podmodely“ tvoří jeden konkrétní model. Model si v tomto kontextu představme například jako automobil. Automobil by se dal rozdělit na části jako karosérie, podvozek a interiér. Každou tuto část bychom zase rozdělili na menší dílčí části. Pro interiér by tyto části mohly být například palubní deska, sedadlo, okna a podobně. Tyto dílčí části se nakonec rozdělí na jednotlivé součástky, které je tvoří (např. volant, přední okno, zadní okno, opěrná část sedadla atd.). Ve Step-file jazyce by byl automobil reprezentován jako celý soubor. **SHELL_BASED_SURFACE_MODEL** by byl typ entity pro karosérii, motor a interiér. **OPEN_SHELL** by pak byly jeho dílčí části jako palubní deska nebo sedadlo. Dále nedělitelné a konečné součástky by pak byly znázorněny entitou **ADVANCED_FACE** (např. volant).

Ukázka zápisu entit:
#12=SHELL_BASED_SURFACE_MODEL('shell_1',(#14));
#14=OPEN_SHELL($,(#15));

5.2.2 Advanced_Face

Tato entita obsahuje odkaz na právě jednu NURBS plochu a také pole „přechodných“ entit, přes které se postupně dostaneme i k ořezovým křivkám pro tuto plochu. Názvy entit, na které **ADVANCED_FACE** odkazuje, jsou **FACE_OUTER_BOUND**, **FACE_BOUND** a **B_SPLINE_SURFACE_WITH_KNOTS**.

Ukázka zápisu entity:
#15=ADVANCED_FACE($,(#18,#16,#17),#40,.T.);

5.2.3 Face_Bound, Edge_Loop, Oriented_Edge a Edge_Curve

Ořezové křivky se dají rozdělit na uzavřené a otevřené. Aby mohla křivka plochu nějakým způsobem ořezat, musí být uzavřená. Uzavřená ořezová křivka je buďto samostatná, nebo se skládá z více otevřených křivek.

FACE_OUTER_BOUND a **FACE_BOUND** jsou entity, které určují právě jeden ořez plochy. Tyto entity jsou provedením stejné a drží odkazy na entity typu **EDGE_LOOP**. Rozdíl mezi těmito dvěmi entitami je ten, že **FACE_OUTER_BOUND** značí ořez vnějšího okraje plochy a **FACE_BOUND** je klasický ořez uvnitř plochy.

Výše zmíněná entita **EDGE_LOOP** obsahuje odkazy na **ORIENTED_EDGE**. Jeden **EDGE_LOOP** může mít odkaz na n-počet křivek, které tvoří jeden ořez.

ORIENTED_EDGE má, kromě odkazu na **EDGE_CURVE**, jeden důležitý atribut, a tím je orientace. Tento boolean atribut může měnit orientaci křivky. Má-li orientace hodnotu true, pak se její směr nemění. Pokud je orientace s hodnotou false, pak je potřeba obrátit načítání kontrolních bodů křivky tak, aby se změnil její směr.

Konečně jako poslední z těchto „přechodných“ entit je zde **EDGE_CURVE**. Tato entita drží odkaz na **B_SPLINE_CURVE_WITH_KNOTS**, který reprezentuje konkrétní výskyt křivky a jejích atributů, a dva odkazy na **VERTEX_POINT**, které jsou pro naše účely nepotřebné, tudíž je nechávám „zaslepené“ a uložené jako string.

Ukázka zápisu entit:
#17=FACE_BOUND($,#21,.T.);
#21=EDGE_LOOP($,(#27));
#27=ORIENTED_EDGE($,*,*,#29,.T.);
#29=EDGE_CURVE($,#39,#39,#49,.T.);

5.2.4 B_Spline_Curve_With_Knots

B_SPLINE_CURVE_WITH_KNOTS je znázornění jedné z ořezových křivek ležících na ploše. Tato entita obsahuje všechny důležité atributy, které NURBS křivka potřebuje pro svou reprezentaci.

Ukázka zápisu entity:
#49=B_SPLINE_CURVE_WITH_KNOTS($,3,(#450,#451,#452,#453,#454,#455),
.UNSPECIFIED., .F.,.F.,(4,1,1,4),(0.,0.341220718234587,0.682441436469174,1.),
.UNSPECIFIED.);

Atributy:

Název	Popis
name	Název křivky. Pokud křivka není pojmenována, pak je implicitní hodnota nastavena na $.
degree	Stupeň křivky p.
control_points	Pole kontrolních bodů. Tyto body jsou typu Cartesian_Point, který bude rozepsaný níže.
curve_form	Forma křivky. Tento atribut není pro tuto práci důležitý.
closed	Atribut, který určuje zdali je křivka uzavřená. Atribut je datového typu bool (true = uzavřená, false = otevřená).
self_intersect	Tento atribut nese informaci zdali křivka kříží samu sebe (true = kříží, false = nekříží).
knot_multi	Pole celých čísel, kde každé číslo značí počet opakování jednoho uzlu v uzlovém vektoru.
knots	Pole desetinných čísel typu double, kde každé číslo reprezentuje jeden uzel v uzlovém vektoru.
spec	Specifikace křivky. Tento atribut není pro tuto práci důležitý.

Tabulka 3: Atributy entity pro křivku

5.2.5 B_Spline_Surface_With_Knots

B_SPLINE_SURFACE_WITH_KNOTS reprezentuje NURBS plochu bez jakéhokoliv ořezu. Tato entita nese všechny informace potřebné ke správnému vykreslení plochy.

Ukázka zápisu entity:
#40=B_SPLINE_SURFACE_WITH_KNOTS($,3,3,((#474,#475,#476,#477,#478,#479),(#480,#481,#482,#483,#484,#485),(#486,#487,#488,#489,#490,#491),(#492,#493,#494,#495,#496,#497),(#498,#499,#500,#501,#502,#503),(#504,#505,#506,#507,#508,#509)),.UNSPECIFIED.,.F.,.F.,.F.,(4,1,1,4),(4,1,1,4),(0.,0.341220718234587,0.682441436469174,1.),(0.,0.348060236457981,0.696120472915962,1.),.UNSPECIFIED.);

Jak je zřejmé z ukázky, tak zápisy plochy a křivky ve formátu STEP se téměř neliší. Oproti křivkám jsou zde dva údaje pro úhly. První z nich je úhel p a druhý q. Za povšimnutí stojí i uspořádání kontrolních bodů v tomto zápisu. Jedná se o dvourozměrné pole, kde jeden rozměr uchovává prvky pro směr u a druhý pro směr v. Posledním rozdílem je uložení uzlových vektorů. Nejprve jsou zde zaznamenány počty multiplicit ve dvou jednorozměrných polích a poté dvě nová pole s jednotlivými hodnotami uzlů.

Atributy:

Název	Popis
name	Název plochy. Pokud plocha není pojmenována, pak je implicitní hodnota nastavena na $.
u_degree	Stupeň plochy p.
v_degree	Stupeň plochy q.
control_points	Dvou rozměrné pole kontrolních bodů pro směr u a v. Tyto body jsou opět typu Cartesian_Point.
surface_form	Forma plochy. Tento atribut není pro tuto práci důležitý.
u_closed	Atribut, který určuje zdali je plocha ve směru u uzavřená.
v_closed	Atribut, který určuje zdali je plocha ve směru v uzavřená.
self_intersect	Tento atribut nese informaci zdali plocha kříží samu sebe.
u_multi	Pole celých čísel, každé číslo značí počet opakování jednoho uzlu v uzlovém vektoru pro směr u.
v_multi	Pole celých čísel, každé číslo značí počet opakování jednoho uzlu v uzlovém vektoru pro směr v.
u_knots	Pole desetinných čísel typu double, kde každé číslo reprezentuje jeden uzel v uzlovém vektoru pro směr u.
v_knots	Pole desetinných čísel typu double, kde každé číslo reprezentuje jeden uzel v uzlovém vektoru pro směr v.
spec	Specifikace plochy. Tento atribut není pro tuto práci důležitý.

Tabulka 4: Atributy entity pro plochu

5.2.6 Cartesian_Point

Pomocí této entity se ve Stepu reprezentuje bod ve 3D prostoru. Jde o základní složku každé křivky a plochy. Všechny hodnoty těchto bodů jsou zadány v modelovém prostoru (z čehož vyplývá pozdější komplikovaný převod do prostoru parametrického) a neobsahují hodnotu váhy. Váha je defaultně nastavena na hodnotu 1. Žádný z těchto bodů se v souboru neopakuje.

Ukázka zápisu entity:
#509=CARTESIAN_POINT($,(50.,50.,-10.));

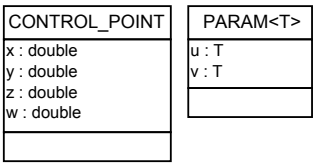
Atributy:

Název	Popis
name	Název bodu. Nemá-li bod žádné jméno, pak je implicitní hodnota nastavena na $.
coordinates	Tříprvkové pole, které obsahuje čísla typu double, kde první prvek má hodnotu osy x, druhý y a třetí z.

Tabulka 5: Atributy entity pro bod

5.3 Converter

V této fázi zpracování dat se převádí mnou vytvořené třídy, reprezentující Step entity, na obecné třídy, se kterými pracuje stávající rozšiřovaná aplikace. K tomuto převodu využívám knihovnu nulib z původního programu, ze které využívám třídy $NurbCurve$, $NurbSurface$ a dále pak pár pomocných datových typů jako $CONTROL\_POINT$ nebo $PARAM\,\langle T\rangle$.

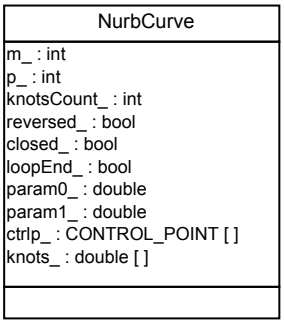


Obrázek 2: UML pro CONTROL_POINT a PARAM

Třída Converter získává pole kořenových instancí tříd Shell_Based_Surface_Model pomocí metody $load$ a vrací převedené pole s objekty typu $NurbSurface$ zavoláním metody $getNurbModels$. V této části programu při převodu modelově zadaných křivek na jejich parametrické vyjádření nastává problém. Jak převést body, kterými je křivka určena, ze 3D (x, y, z) do parametrického vyjádření pomocí (u, v)? Řešení tohoto problému proberu podrobněji v kapitole 5.3.3.

5.3.1 NurbCurve

Tato třída reprezentuje obdobné vyjádření NURBS křivky, podobně jako předchozí třída B_Spline_Curve_With_Knots. Rozdílem mezi těmito třídami je použití nových datových typů, změna názvů proměnných a vůbec kompletní změna hlavičkové sekce. Tato nová třída rovněž obsahuje nové metody vhodné pro normalizaci kontrolních bodů a uzlového vektoru, nebo pro textový výpis hodnot všech atributů křivky.



Obrázek 3: UML diagram pro NurbCurve

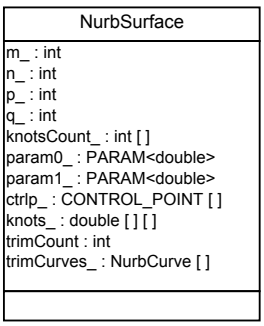
Význam atributů třídy NurbCurve:

Název	Popis
m_	Počet kontrolních bodů křivky.
p_	Stupeň křivky.
knotsCount_	Počet uzlů v uzlovém vektoru.
reversed_	Bool hodnota, která určuje zda křivka řeže nebo zachovává ohraničenou část plochy.
closed_	Bool hodnota, která určuje zda je křivka uzavřená nebo otevřená.
loopEnd_	Atribut, který nám říká, jestli se jedná o poslední segment v poli křivek tvořících uzavřenou smyčku.
param0_, param1_	Pomocné parametry využívané při normalizaci křivky.
ctrlp_	Pole kontrolních bodů, které určují tvar křivky.
knots_	Uzlový vektor, ve kterém jsou uloženy všechny uzly i s povolenými duplicitami.

Tabulka 6: Atributy třídy NurbCurve

5.3.2 NurbSurface

Třídá NurbSurface reprezentuje NURBS plochu podobným způsobem jako instance entity B_Spline_Surface_With_Knots. Tato třída obsahuje všechny atributy, které jsou důležité pro správné vykreslení plochy a navíc také pole, které v sobě uchovává všechny ořezové křivky této plochy. Stejně jako předchozí třída má i tato nové metody určené pro normalizaci a výpis. Díky poli ořezových křivek je zde také pár nových metod určených pro správu tohoto pole. V průběhu samotné aplikace je tato třída stěžejní a v části věnované vykreslování se už pracuje pouze s ní (nikoliv s entitou souboru STEP). Tato třída je rovněž důležitým mezikusem při budoucím zkombinování této rozšiřující aplikace s původním programem.



Obrázek 4: UML diagram pro NurbSurface

Význam atributů třídy NurbSurface:

Název	Popis
m_	Počet kontrolních bodů plochy ve směru u.
n_	Počet kontrolních bodů plochy ve směru v.
p_	Stupeň plochy ve směru u.
q_	Stupeň plochy ve směru v.
knotsCount_	Dvouprvkové pole, kde první hodnota určuje počet uzlů pro směr u a druhá pro počet uzlů pro směr v.
param0_, param1_	Pomocné parametry využívané při normalizaci křivky. Tyto parametry jsou typu PARAM, který je popsaný výše (obrázek 5.3).
ctrlp_	Pole kontrolních bodů, které určují tvar křivky.
knots_	Dvou rozměrné pole. V prvním rozměru jsou pouze dva prvky, jeden pro směr u a druhý pro v. Další rozměr obsahuje uzly pro svůj směr a má velikost stejnou, jako je hodnota pro daný směr v atributu knotsCount.
trimCount_	Počet ořezových křivek na ploše.
trimCurves_	Pole všech ořezových křivek plochy.

Tabulka 7: Atributy třídy NurbSurface

5.3.3 Point Inversion

Mějme bod $P = (x, y, z)$ a předpokládejme, že leží na NURBS ploše $S(u, v)$ se stupni p a q. Inverze bodu (point inversion) je činnost, která řeší vyhledání korespondujících parametrů $\overline{u}$ a $\overline{v}$, kde $S(\overline{u}, \overline{v}) = P$.
Kroky pro převod bodu:

- Vyjmutím všech rozsahů (spans) mezi sousedícími uzly z uzlových vektorů definujeme množiny, které využijeme při hledání P.
- Ze středů takto získaných rozsahů pro oba směry vytvoříme v parametrickém prostoru mřížku, kde každý bod tvořený touto mřížkou bude parametrickým kandidátem na bod modelového prostoru.
- Tyto kandidáty pak testujeme pomocí níže popsaných algoritmů zdali odpovídají (popřípadě se blíží) některému zadanému bodu.

Metoda, kterou využijeme pro kontrolu takto získaných bodů, se nazývá Newtonova iterační metoda. Tato metoda snižuje nepřesnosti tak, že minimalizuje vzdálenost mezi bodem P a plochou $S(u, v)$. Předpokládá se, že bod P leží na ploše, pokud je jeho vzdálenost od ní menší než zadaná tolerance epsilon ϵ.

Programové zpracování konkrétních výpočtů pro bodovou inverzi je obsaženo v includované knihovně z původní rozšiřované aplikace. Ve třídě Convertor tento přepočet využívám pomocí metody $pointInversion$, do které vkládám vhodné kandidáty a testuji zdali jsou výsledné parametry s povolenou odchylkou. Pokud nejsem schopen nalézt vhodného kandidáta na bod s určenou odchylkou, snížím přesnost hledání a provedu jej znova. Tímto způsobem hledám body až do okem viditelné nepřesnosti. Pokud nejsem schopen nalézt kandidáty ani po určité době snižování přesnosti, ukončím výpočty a výslednou plochu vykreslím bez ořezů, abych se vyhnul časové náročnosti programu.

Kvůli časové náročnosti a nepřesnosti výsledků se takovéto řešení v praxi moc nevyužívá. Efektivnější programy jako Rhinoceros nebo Maya pravděpodobně využívají jiných algoritmů a procedur, které bohužel nejsou open source. Tyto profesionální aplikace navíc většinu svých grafických propočtů provádí úplně jiným způsobem s širší škálou možností.

5.4 Přepočty a normalizace

V této části bych se rád podrobněji věnoval důležitým výpočtům, které aplikace provádí před vykreslením objektů na scénu. Většina přepočtů se provádí zároveň při převodu Step tříd na obecné třídy v minulé části 5.3.

Jak bylo řečeno v části 2.1.2 věnované uzlovému vektoru, tak normalizace NURBS poloch a křivek není povinná. Zápis NURBS s normalizovanými hodnotami je ekvivalentní zápisu bez normalizace. Normalizace hodnot před samotnými výpočty je však obecným zvykem, a tudíž je vhodné ji provést. V pozdější části programu se navíc před vykreslením provede renormalizace hodnot, aby se odstranila nesourodost výsledků, která mohla nastat při menších úpravách a převodech z double na float.

5.4.1 Normalizace uzlového vektoru

Normalizace uzlového vektoru křivky se provádí jednoduchým způsobem. Předpokládejme správnost zadání vektoru (tzn. má neklesající posloupnost hodnot s nepřekročenou hranici multiplicit), díky čemuž máme jistotu, že první uzel v tomto vektoru má nejmenší možnou hodnotu a poslední uzel má naopak hodnotu největší. Rozdílem hodnot těchto dvou krajních uzlů získame normalizační koeficient, kterým následně dělíme výsledný rozdíl mezi stávajícím a prvním uzlem vektoru.

```
void NurbCurve::normalizeKnotVector()
{
    param0_ = knots_[0];
    param1_ = knots_[knotsCount_-1];
    double norm = param1_ - param0_;
    for ( int i = 0; i < knotsCount_; i++ ) {
        knots_[i] = ( knots_[i] - param0_ ) / norm;
    }

    //nastavení nových fixních mezí
    param0_ = 0.0;
```

```
    param1_ = 1.0;
}
```

Výpis 5: Metoda pro normalizaci uzlového vektoru křivky

Podobným způsobem jako u křivek se normalizují oba vektory u NURBS ploch. Jediným rozdílem je, že každý vektor plochy se normalizuje „sám za sebe“. Parametry plochy jsou datového typu $PARAM\,\langle double\rangle$, kde každý z nich obsahuje hodnotu pro směr u i v. Normalizace je pak odlišná jen v tom, že se počítá normalizační koeficient pro oba směry zvlášť. Na parametrech získaných z původních vektorů později funguje i normalizace kontrolních bodů křivky, která je probrána v odstavci 5.4.2 níže.

```
void NurbSurface::normalizeKnotVectors()
{
    //pro směr u
    double norm = param1_.u − param0_.u;
    for ( int i = 0; i < knotsCount_[U_DIR]; i++ ) {
        knots_[U_DIR][i] = ( knots_[U_DIR][i] − param0_.u ) / norm;
    }
    //pro směr v
    norm = param1_.v − param0_.v;
    for ( int i = 0; i < knotsCount_[V_DIR]; i++ ) {
        knots_[V_DIR][i] = ( knots_[V_DIR][i] − param0_.v ) / norm;
    }

}
```

Výpis 6: Metoda pro normalizaci uzlových vektorů plochy

5.4.2 Normalizace kontrolních bodů

V této části normalizace již uvažujme, že kontrolní body určující tvar křivky, byly převedeny z modelového prostoru do prostoru parametrického. Díky tomuto převodu již body nemají tvar $P = (x, y, z)$, ale $P = (u, v, w)$, kde w je váha bodu (nejčastěji hodnoty 1). Jak bylo řečeno v předchozí kapitole 5.4.1, k normalizaci kontrolních bodů křivky je potřeba znát parametry plochy, na které křivka leží. Tyto parametry zaručí, že se křivka znormalizuje vůči ploše, kterou ořezává, a díky tomu zůstanou zachovány poměry velikostí ořezů oproti neznormalizované verzi.

```
void NurbCurve::normalizeCtrlp( PARAM<double> & param0, PARAM<double> & param1 )
{
    double normu = param1.u − param0.u;
    double normv = param1.v − param0.v;

    for ( int i = 0; i < m_; i++ ) {
        CONTROL_POINT & point = ctrlp_[i];

        point.x = ( point.x − param0.u ) / normu;
```

```
            point.y = ( point.y − param0.v ) / normv;
            point.z = 0.0;
            // w zustava stejne
        }
    }
```

Výpis 7: Metoda pro normalizaci kontrolních bodů křivky

5.5 Hlavní třída NuVis_classes a metoda main

V této části aplikace se definuje vzhled a vlastnosti okna pro finální vykreslení. Rovněž jsou zde implementovány všechny funkce potřebné pro ovládání aplikace nebo pro manipulaci s objekty ve scéně.

5.5.1 NuVis_classes

V této třídě je obecně implementováno nastavení okna, do kterého se v pozdější fázi bude vykreslovat scéna. Jsou zde nastavení, jako například velikost okna, vlastnosti a barva materiálu a všechny funkce pro ovládání myší i klávesnicí. Ovládání aplikace je přesně popsáno v kapitole 6.

V této části programu se také nachází různé pomocné funkce, jednou z nich je například vykreslování řídící mřížky, která naznačuje osy ve scéně. Rovněž je zde umístěno samotné vykreslení scény. Scéna se vykresluje pomocí metod implementovaných ve třídě Universe. Při prvním vykreslení scény se volá metoda $preDraw$, která zajistí inicializaci display listů a do nich následně uloží všechny objekty ve scéně. V dalším překreslování scény při posunu nebo rotaci kamery se pak volá pouze metoda $draw$, která už jen vyvolává uložené objekty z výše zmiňovaných display listů.

5.5.2 Metoda main

V těle této metody se pomocí argumentu přidaného při spuštění aplikace rozhoduje, zdali se jedná o validní soubor, který by bylo možno později vykreslit. Validním souborem máme na mysli takový, který má příponu $*.stp$ nebo $*.cba$. Určením přípony vstupního souboru se rozhodne jaké načítací metody se vyvolají. Nastane-li při načítání chyba, bude vyvolána odpovídající varovná hláška a program bude ukončen.

5.6 Vykreslení universa

Universem vnímáme soubor všech objektů ve scéně. Pro manipulaci s těmito objekty byla vytvořena pomocná třída Model, která je podrobněji popsána v následující sekci 5.6.1. Universe obsahuje pole instancí třídy Model, metody pro práci s tímto polem, metodu pro „předkreslení“ všech modelů do display listů a metodu pro vykreslení všech modelů vyvolaných z display listů.

Display listy si můžeme představit jako makra, do kterých se nahraje několik příkazů OpenGL, a tato makra lze potom jedním příkazem „spustit“. Výhodou display listů je na

jedné straně zvýšená rychlost vykreslování, protože display listy jsou většinou uloženy přímo v paměti grafického akcelerátoru, na straně druhé také zjednodušení kódu pro vykreslování komplikovaných scén. Je například možné 3D modely jednotlivých těles ukládat do samostatných display listů a složitou scénu potom vykreslit pouze zavoláním těchto display listů s vhodně nastavenou transformační maticí.

Po rozhodnutí o jaký typ vstupního souboru se jedná, se v universu zavolá patřičná načítací metoda. V universu jsou implementovány dva druhy metod vhodných pro načítání dat ze vstupního souboru. První metoda $loadStep$ pracuje se souborem $*.stp$, zatímco druhá $loadCba$ pracuje s daty získanými ze souboru $*.cba$.

V první metodě vytvořené pro načítání ze souborů $*.stp$ se vyvolá nová instance třídy Parser, která vytvoří pole s adresovými klíči kořenových prvků modelové struktury a mapu všech prvků v této struktuře. Po získání této mapy a kořenových klíčů máme všechny důležité informace potřebné k vytvoření pole všech modelových struktur. Tyto struktury se následně převedou pomocí instance třídy Converter na pole s prvky typu NurbSurface. Výsledné pole se uloží do atributu universa a je připraveno pro vykreslení.

Druhá metoda určená pro načítání universa ze souboru $*.cba$ má konstrukci jednodušší. Díky načítací metodě použité z inkludované knihovny nulib.dll se načítání dat provede už v metodě main. Samotné načtení metodou universa je pak jen přeposlání načteného pole prvků typu NurbSurface. Díky tomuto pak odpadá veškeré parsování a převody různých datových typů. Po uložení takto načteného pole do atributu universa je vše připraveno pro vykreslení scény.

5.6.1 Pomocná třída Model

Třída Model uchovává nejen konkrétní objekt ve scéně, ale zároveň implementuje metody pro jeho vykreslení (respektive uložení do display listů) a renormalizaci.

Před samotným vykreslením plochy je důležité provést kontrolu hodnot. V průběhu zpracovávání dat a normalizací se může například stát, že hodnoty uzlů v uzlovém vektoru nabydou nechtěných multiplicit mimo povolenou hranici. Proto se v této části aplikace provádí úprava uzlových vektorů pomocí metody $fixKnot$. Tato metoda postupně prochází všechny uzly kontrolovaného vektoru, a pokud narazí na uzel s multiplicitou vyšší než je stupeň křivky, tak provede jeho úpravu. Úprava multiplicitních hodnot spočívá v mírném posunu daného uzlu a zároveň všech uzlů následujících. Důvod, proč se provádí posun všech následujících uzlů, je ten, že bez této úpravy by se některé uzly mohly „přeskočit“, a tím by byly porušeny podmínky pro správnost zadání uzlového vektoru (porušení neklesající posloupnosti). Po této úpravě uzlového vektoru je potřeba jej znovu normalizovat do intervalu $\langle 0, 1 \rangle$. V této třídě se dále provádí další důležitá úprava, bez které by se vykreslování ploch nedalo provést. V předchozích částech této práce bylo několikrát zmíněno, že plochu je možno oříznout pouze pomocí uzavřených křivek.Tato úprava by se dala nazvat „svařování křivek“. Svařování se provádí pomocí metody $getRegions$, která vrací pole, ve kterém jsou uloženy jednotlivé regiony naplněné křivkami. Tyto křivky jsou v regionech seřazeny tak, jak na sebe postupně navazují koncovými body. Díky tomuto seřazení se dá n oteřených křivek číst jako jedna uzavřená ořezová NURBS křivka.

6 Ovládání aplikace

Před samotným spuštěním je důležité mít v adresáři, ve kterém je umístěn soubor NuVis.exe, vložen soubor glut32.dll. Bez této přiložené knihovny a knihovny GLU s verzí alespoň 1.3 nelze očekávat od aplikace NuVis správnou funkčnost.

Možnosti spuštění této aplikace jsou dvojí. Program NuVis může příjmout vstupní soubor jako parametr při spouštění z příkazové řádky, nebo je možné soubor tažením přenést na spouštěcí ikonu aplikace (obdoba „Otevřít v programu...“). Po kontrole vstupního souboru a načtení dat se na příkazové řádce vypíše popis ovládání a zobrazí se okno s vykreslenou scénou.

6.1 Ovládání pomocí klávesnice

Při otevřeném okně je k dispozici velká škála rychlých kláves, které usnadní práci s aplikací, nebo vyvolají informace o zobrazované scéně.
Klávesy:

- **H** — Vypíše na příkazovou řádku informace o programu a jeho ovládání.
- **Esc** — Ukončí běh aplikace.
- **Q** — Podobně jako klávesa Esc ukončí běh aplikace.
- **F** — Přepne okno do fullscreen módu.
- **W** — Vrátí velikost okna na původní hodnoty.
- **G** — Zapíná a vypíná zobrazení řídící mřížky.
- **P** — Vypíše na příkazovou řádku informace o zobrazované ploše.
- **T** — Vypíše na příkazovou řádku informace o zobrazované ploše a jejích ořezech.
- **C** — Vyčistí příkazovou řádku.

6.2 Ovládání pomocí myši

Ovládání pomocí myší umožňuje uživateli mnohem lepší a instinktivnější interakci s vykreslenou scénou.
Držením tlačítka na myši a tažením se dá scéna ovládat těmito způsoby:

- **Pravé tlačítko** — Držením tohoto tlačítka a posunem po vertikální ose posunuje kameru k nebo od středu scény.
- **Levé tlačítko** — Stisk tohoto tlačítka a posun po obou osách rotuje kameru kolem středu vykreslené scény.

7 Závěr

Cílem této práce bylo seznámit se se základními důležitými pojmy jako NURBS, STEP, OpenGL nebo GLU a umět je využít při tvorbě požadované aplikace. Výsledná aplikace je schopna příjmout předem definované vstupní soubory, které posléze rozparsuje na požadované entity. Soubory, které aplikace příjmá jsou formátu STEP nebo CBA. Vstupní formát CBA byl přidán v průběhu práce na aplikaci, aby bylo možné porovnat přesnosti modelů. Tato kontrola přesnosti vychází z nutnosti přepočtů z modelového prostoru do parametrického. Entity vyparsované ze vstupních souborů jsou provázány stejně jako v souboru formátu STEP a dále jsou přepočteny a zpracovány do finální podoby ve formě instancí tříd reprezentujicích NURBS plochy a jejich ořezové křivky. Tyto instance jsou dále překontrolovány a vykresleny jako scéna v OpenGL. Tato scéna má přesně definované vlastnosti a ovládání, které bylo popsáno v posledních částech práce.

Konečná aplikace slouží jako testovací platforma pro implementovaný parser. Samotná práce parseru byla naprogramována co nejefektivněji s co nejmenšími nároky na místo v paměti. Tento parser zachovává hierarchickou strukturu získanou ze souboru STEP. Pro kontrolu těchto vyparsovaných NURBS ploch je zde vytvořen vizualizátor, který pomocí OpenGL vykreslí výslednou scénu. Budoucí možné úpravy aplikace mohou být: tvorba rozšířenějšího uživatelského rozhraní, přidání vstupů pro nové typy souborů a úprava některých algoritmů pro urychlení výpočtů.

V průběhu práce na této aplikaci a její dokumentaci jsem objevil spoustu možností, které NURBS plochy spolu s knihovnou OpenGL nabízí. Budoucnost takto reprezentovaných ploch je podle mého názoru slibná hlavně díky tomu , že se jedná o rychlé a přesné výpočty při jejich tvorbě. S rostoucími nároky firem, které ve svých projektech využívají tuto technologii, jde ruku v ruce náročnost pro výpočty takto tvořených modelů. Proto je důležité stále vyvíjet výpočetní metody a algoritmy z důvodů jejich vyšší efektivity.

Zároveň jsem zjistil, že souborový formát STEP je velice zajímavým, a po jeho pochopení také přehledným záznamem dat. Takto reprezentované soubory jsou s určitou praxí dobře čitelné a dá se říci, že díky jejich stavbě i jednoduše transformovatelné na vlastní hierarchické struktury, nebo (při vhodném návrhu) soubory typu XML.

8 Reference

[1] PIEGL, Les; TILLER, Wayne. *The NURBS Book*. New York : Springer, 1996. xiv, 646 s.

[2] ANGEL, Edward. *Interactive Computer Graphics*. United States of America : Pearson Education, 2009. xxix, 828s s.

[3] *Step Application Handbook ISO10303* [online]. North Charleston : SCRA, 30 June 2006 [cit. 2010-04-22]. Dostupné z WWW: <http://www.isg-scra.org/files/STEP_Application_Handbook.pdf>.

[4] PROCHÁZKOVÁ, Jana. *Root.cz* [online]. 2006-03-03 [cit. 2010-04-20]. Křivky NURBS (1). Dostupné z WWW: <http://www.root.cz/clanky/krivky-nurbs-1/>

[5] PROCHÁZKOVÁ, Jana. *Root.cz* [online]. 2006-03-10 [cit. 2010-04-20]. Křivky NURBS (2). Dostupné z WWW: <http://www.root.cz/clanky/krivky-nurbs-2/>

[6] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2003-08-19 [cit. 2010-04-20]. Grafická knihovna OpenGL (8): display-listy. Dostupné z WWW: <http://www.root.cz/clanky/opengl-8-display-listy/>

[7] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2008-02-21 [cit. 2010-04-20]. Formát X3D a záhadná zkratka NURBS. Dostupné z WWW: <http://www.root.cz/clanky/format-x3d-a-zahadna-zkratka-nurbs/>

[8] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2003-02-28 [cit. 2010-04-20]. NURB křivky a plochy ve formátu X3D. Dostupné z WWW: <http://www.root.cz/clanky/nurb-krivky-a-plochy-ve-formatu-x3d/>

[9] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2004-09-21 [cit. 2010-04-20]. OpenGL a nadstavbová knihovna GLU (9). Dostupné z WWW: <http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu-9/>

[10] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2004-10-05 [cit. 2010-04-20]. OpenGL a nadstavbová knihovna GLU (10). Dostupné z WWW: <http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu-10/>

[11] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2004-10-12 [cit. 2010-04-20]. OpenGL a nadstavbová knihovna GLU (11). Dostupné z WWW: <http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu-11/>

[12] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2004-10-26 [cit. 2010-04-20]. OpenGL a nadstavbová knihovna GLU (13). Dostupné z WWW: <http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu-13/>

[13] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2004-11-09 [cit. 2010-04-20]. OpenGL a nadstavbová knihovna GLU (15). Dostupné z WWW: <http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu-15/>

[14] TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 2004-11-16 [cit. 2010-04-20]. OpenGL a nadstavbová knihovna GLU (16). Dostupné z WWW: <http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu-16/>

[15] Non-uniform rational B-spline In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , [cit. 2010-04-20]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline/>.

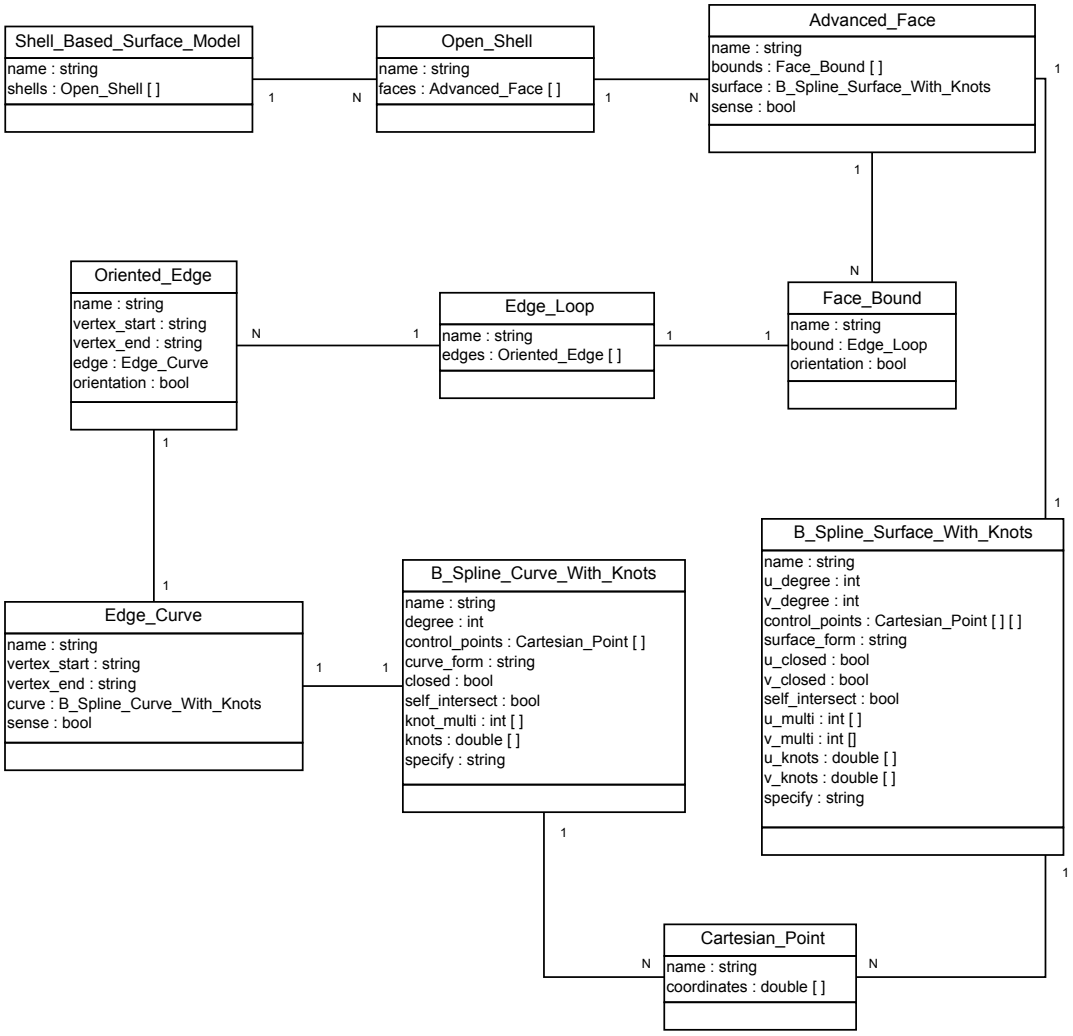
[16] Freeform surface In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , [cit. 2010-04-20]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Freeform_surface>.

[17] OpenGL In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , [cit. 2010-04-23]. Dostupné z WWW: <http://en.wikipedia.org/wiki/OpenGL>.

[18] ISO 10303 In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , [cit. 2010-04-23]. Dostupné z WWW: <http://en.wikipedia.org/wiki/ISO_10303>.

A UML struktura

Tato struktura popisuje provázání jednotlivých prvků tvořících NURBS plochu a její ořezové křivky ve formátu Step.



Obrázek 5: Ukázka struktury entit

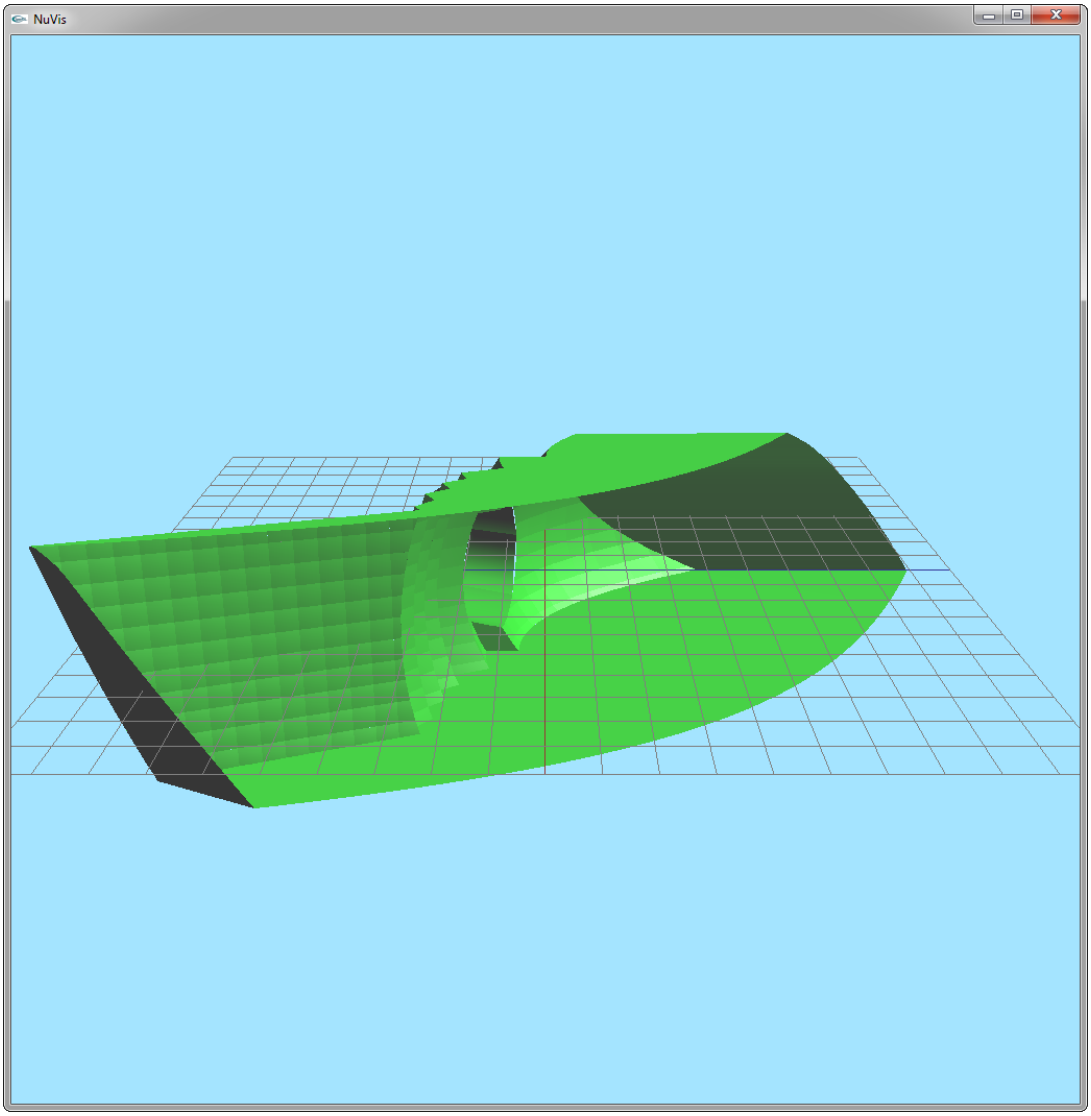
B Ukázka souboru formátu Step

```
ISO-10303-21;
HEADER;
/* Generated by software containing ST-Developer
 * from STEP Tools, Inc. (www.steptools.com)
 */
/* OPTION: using custom schema-name function */

FILE_DESCRIPTION(
/* description */ (''),
/* implementation_level */ '2;1');

FILE_NAME(
/* name */'example_trim4',
/* time_stamp */'2009-11-10T14:13:14+01:00',
/* author */ (''),
/* organization */ (''),
/* preprocessor_version */ 'ST-DEVELOPER v10',
/* originating_system */ '',
/* authorisation */ '');

FILE_SCHEMA (('CONFIG_CONTROL_DESIGN'));
ENDSEC;

DATA;
#12=SHELL_BASED_SURFACE_MODEL('shell_1',(#14));
#14=OPEN_SHELL($,(#15));
#15=ADVANCED_FACE($,(#18,#16,#17),#40,.T.);
#17=FACE_BOUND($,#21,.T.);
#21=EDGE_LOOP($,(#27));
#27=ORIENTED_EDGE($,*,*,#29,.T.);
#29=EDGE_CURVE($,#39,#39,#49,.T.);
#39=VERTEX_POINT($,#515);
#40=B_SPLINE_SURFACE_WITH_KNOTS($,3,3,((#474,#475,#476,#477,#478,#479),(#480,
#481,#482,#483,#484,#485),(#486,#487,#488,#489,#490,#491),(#492,#493,#494,
#495,#496,#497),(#498,#499,#500,#501,#502,#503),(#504,#505,#506,#507,#508,
#509)),.UNSPECIFIED.,.F.,.F.,.F.,(4,1,1,4) ,(4,1,1,4) ,(0.,0.341220718234587,
0.682441436469174,1.),(0.,0.348060236457981,0.696120472915962,1.),.UNSPECIFIED.);
#49=B_SPLINE_CURVE_WITH_KNOTS($,3,(#450,#451,#452,#453,#454,#455),.UNSPECIFIED.,
.F .,. F .,(4,1,1,4) ,(0.,0.341220718234587,0.682441436469174,1.),.UNSPECIFIED.);
#50=B_SPLINE_CURVE_WITH_KNOTS($,3,(#456,#457,#458,#459,#460,#461),.UNSPECIFIED.,
.F .,. F .,(4,1,1,4) ,(0.,0.348060236457981,0.696120472915962,1.),.UNSPECIFIED.);
#142=SHAPE_REPRESENTATION('Document',(#143,#144,#145),#141);
#143=AXIS2_PLACEMENT_3D('',#152,#146,#147);
#146=DIRECTION($,(0.,0.,1.));
#152=CARTESIAN_POINT($,(0.,0.,0.));
#154=CARTESIAN_POINT($,(-22.4508896521856,8.41910163504858,7.36176868690551));
ENDSEC;END-ISO-10303-21;
```

Výpis 8: Výtažek ze soubor Step

C Ukázka

Zde je ukázka vykreslení komplexního modelu, jenž tvoří přes 350 ploch.



Obrázek 6: Ukázka vykreslení modelu